UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P O Box 1450
Alexandria, Virginia 22313-1450
www uspto gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/676,373	09/30/2003	Stefan Jesse	09700.0216-00	3224

60668 7590 06/02/2009
SAP / FINNEGAN, HENDERSON LLP
901 NEW YORK AVENUE, NW
WASHINGTON, DC 20001-4413

EXAMINER	
VU, TUAN A	

ART UNIT	PAPER NUMBER
2193	

MAIL DATE	DELIVERY MODE
06/02/2009	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	**Application No.** 10/676,373	**Applicant(s)** JESSE ET AL.
	Examiner TUAN A. VU	**Art Unit** 2193

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on 20 March 2009.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) 1-4,6-12 and 14-24 is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) 1-4,6-12 and 14-24 is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)	4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)	5) ☐ Notice of Informal Patent Application
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.	6) ☐ Other: _____.

DETAILED ACTION

1. This action is responsive to the Applicant's response filed 3/20/09.

Per the decision by the Pre-appeal conference in view of Applicant's response, the Finality of the previous Office Action has been withdrawn. Claims 1-4, 6-12, 14-24 are pending for prosecution as presented in the following Action.

Double Patenting

2. The nonstatutory double patenting rejection is based on a judicially created doctrine grounded in public policy (a policy reflected in the statute) so as to prevent the unjustified or improper timewise extension of the "right to exclude" granted by a patent and to prevent possible harassment by multiple assignees. A nonstatutory obviousness-type double patenting rejection is appropriate where the conflicting claims are not identical, but at least one examined application claim is not patentably distinct from the reference claim(s) because the examined application claim is either anticipated by, or would have been obvious over, the reference claim(s). See, e.g., *In re Berg*, 140 F.3d 1428, 46 USPQ2d 1226 (Fed. Cir. 1998); *In re Goodman*, 11 F.3d 1046, 29 USPQ2d 2010 (Fed. Cir. 1993); *In re Longi*, 759 F.2d 887, 225 USPQ 645 (Fed. Cir. 1985); *In re Van Ornum*, 686 F.2d 937, 214 USPQ 761 (CCPA 1982); *In re Vogel*, 422 F.2d 438, 164 USPQ 619 (CCPA 1970); and *In re Thorington*, 418 F.2d 528, 163 USPQ 644 (CCPA 1969).

A timely filed terminal disclaimer in compliance with 37 CFR 1.321(c) or 1.321(d) may be used to overcome an actual or provisional rejection based on a nonstatutory double patenting ground provided the conflicting application or patent either is shown to be commonly owned with this application, or claims an invention made as a result of activities undertaken within the scope of a joint research agreement.

Effective January 1, 1994, a registered attorney or agent of record may sign a terminal disclaimer. A terminal disclaimer signed by the assignee must fully comply with 37 CFR 3.73(b).

3. Claims 8, 18 are provisionally rejected on the ground of nonstatutory obviousness-type double patenting as being unpatentable over claims 4, 12, 19 of copending Application No. 10,676,374 (hereinafter '374).

Although the conflicting claims are not identical, they are not patentably distinct from each other because of the following example of conflicting claims.

As per instant claim 8, copending '374 claim 4 recites a first data model being used to derive an API and employing the API to access development objects. But '374 claim 4 does not explicitly recite (i) first model defining development objects as building blocks for the application; (ii) generate intermediate objects therefrom, intermediate objects comprising Java objects, and (iii) using the set of intermediate objects as inputs to generate the API, the model including (iv) a first model customized extension used to implement a feature of the API such as an indication of a file border and (v) API enforces relationships and constraints defined in the first model.

However, '374 claim 4 recites a variation of the language in claim 8 for limitations (i) and (ii) via the recital of 'defining file borders comprising identifying of development objects to be included in a file … in the data model … associated component class … to be children of the main …object that are not identified as main…objects', the intermediate objects being added objects to the file of the main object including development objects being OO classes defined from the data model; rendering the teaching in claim 4 obvious language variation of (i) and (ii). As for the *constraints enforcing* limitation of (v) based (iii)-- *using the set of intermediate objects as input for the API generating*—'374 claim 4 includes file storing user-defined code associated with the main development object; and in view '374 teaching of a definition file (see '374 claim 4) in light of objects being defined -- in terms of parent/child relationship in '374 claim 3 – one of ordinary skill would recognize these defined objects as well-known interrelated model components viewed in a GUI development interface of '374, i.e. API - to necessarily support user's development via instantiating one such development GUI API for accessing model objects. And this as a whole would be equivalent to (iii) for enforcing constraints as suggested above, or otherwise obvious variation thereof. As for the *customizable extension* comprising a file border indication referred to as (iv), this is suggested in '374 reciting of 'defining file borders for development', and storing development objects in a repository based on the file borders, and accessing these objects via the API (*); so that one skill in the art would be motivated to provide an extension structure obtained from the repository (e.g. template builder) in the course of the API derivation with utilizing of information in the '374 stored file-based repository for the derivation. That is, the information thus extended (e.g. via a template builder) from the stored model/repository regarding a particular file border identity would be used to support the creation of API parameter or attributes which would be needed to access the very components stored from the '374 defining of file borders, as purported by the endeavor described as (*) from the above.

As per instant claim 8, '374 claim 19 also recites API derived from a data model, file borders defined in the model, and user interface using the API to access development objects being stored in a repository. Claim 19 does not recite 'first language model with extension to implement API and file border; but based on '374 reciting of association between component and model class 'that associates a user interface to a ... application model', the extension by use of border file suggest the *extension* limitation of instant claim 8 to provide deriving of association between stored model objects; that is, obviousness in terms of instant claim 8 limitations such as API for 'enforcing constraints' and 'language extension' does apply here in view of the rationale set forth above.

As per instant claim 18, '374 claim 12 also recites an obvious language variant thereof in expressing 'receiving of a model' in a development method, a *language extension* for defining a model representing blocks in terms of 'component class' and 'model class' as well as their inter-association for developing an application (Note: this would be an obvious variation of building block relationships among objects and, hence suggesting constraints thereof), deriving a API based thereon, and use the API for *enforcing constraints* in the model within the development of the application.

This is a provisional obviousness-type double patenting rejection because the conflicting claims have not in fact been patented.

Claim Rejections - 35 USC § 102

4. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

5. Claims 10, 14 are rejected under 35 U.S.C. 102(e) as being anticipated by Kadel et al, USPubN: 2002/0184401 (hereinafter Kadel).

As per claim 10, Kadel discloses a computer program product, tangibly embodied in an information carrier, for developing an application, the computer program product being operable to cause data processing apparatus to:

receive a first model in a first language from a storage device, the first model defining development objects representing building blocks for developing the application, *relationships among the development objects* (e.g. Fig. 3B; para 0117-0126, pg. 8-9 – Note: XIS framework – see Fig. 3A - using InfoModel or InfoBean model paradigm and graphical representation accessible to user and -- para 0094, pg. 6; Domains, policies, relationships - para 0096-0097, pg. 7-- represented in blocks or sequence diagrams, or domain relationships in relation to consumer/producer interaction – see Fig. 13; UML - see Fig. 8, 11 - reads on first model in first language), *and constraints for developing the application* (e.g. listener, interrogate, RemovedEvent, ReferenceEvent - Fig. 36A-D; Fig. 11; para 0153-0156, pg.11; 0167, pg. 12; value constraints - para 0178, pg. 13; attribute constraints – para 0320 pg. 27), wherein the first language comprises unified modeling language (e.g. Fig. 8, 11, 13, 36C-D);

generate a set of intermediate objects using the first model (Fig. 5; para 0194-0198, pg. 15; *JavaBeans ... package com.xis.leif.event ... includes interfaces* - para 0210-0213, pg. 16; *domain policy methods ... returned ... procedural components of the metadata and methods* - para 0334, pg. 29); and

generate an XML schema such that the XML schema enforces the relationships and the constraints defined in the first model and enables implementing the development objects (e.g. attribute constraints – para 0320, pg. 27; para 0288-0304, pg. 24; para 0083-0085, pg. 5; *XML schema ... sends to the receiver ... receiver reconstructs the information* - para 0305, pg. 25; para 0318, pg. 26; Fig. 30 – Note: importing of XML schema to reconstruct application reads on schema with meta-information enforcing a relationship and constraints – see Kadel: para 0153-0156, pg.11; 0167, pg. 12; value constraints - para 0178, pg. 13)

Kadel discloses JFC (0163, pg. 11) J2EE packages (para 0311-0312, pg. 26) and Java interface packages (Fig. 12A-D) to support representation of UML constructs (see above) and the XIS framework enabling user editing and manipulation of metadata exposed by the DSI (Fig. 2-4); e.g. as to implement a source/consumer paradigm (Fig. 11, 13) in view of the UML sequence

(Fig. 13) and the retrieved meta attributes for a specific domain properties being exposed (Fig. 5). Further, Kadel discloses converting of metadata attributes of a XIS/DSI database in terms of creation of extensible schema (see para 0309-0320, pg. 25-27), the XML language created a as to representing attribute derived from the DSI exposure process and correlate its markup language with Java code being embedded (see para 0316-0320, pg. 27); hence Kadel has disclosed generating XML schema **using information input** such as the *set of intermediate objects being Java classes* as shown above.

As per claim 14, Kadel discloses wherein the set of intermediate objects comprises Java objects (refer to claim 10).

Claim Rejections - 35 USC § 103

6. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

7. Claims 1, 4, 6-9, 18, 23-24 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kadel et al, USPubN: 2002/0184401 (hereinafter Kadel), in view of Hu, USPubN: 2003/0196168 (hereinafter Hu), and further in view of Severin, USPubN: 2005/0005251 (hereinafter Severin).

As per claim 1, Kadel discloses a computer program product, tangibly embodied in an information carrier, for developing an application, the computer program product being operable to cause data processing apparatus to:

receive a first model in a first language, the first model defining development objects representing building blocks for developing the application, relationships among the development objects, (e.g. Fig. 3B; para 0117-0126, pg. 8-9 – Note: XIS framework – see Fig. 3A - using InfoModel or InfoBean model paradigm and graphical representation accessible to user and -- para 0094, pg. 6; Domains, policies, relationships - para 0096-0097, pg. 7-- represented in blocks or sequence diagrams, or domain relationships in relation to consumer/producer interaction – see Fig. 13; UML - see Fig. 8, 11 - reads on first model in first language), and constraints for developing the application (e.g. listener, interrogate, RemovedEvent, ReferenceEvent - Fig. 36A-D; Fig. 11; para 0153-0156, pg.11; 0167, pg. 12; value constraints - para 0178, pg. 13; attribute constraints – para 0320 pg. 27);

generate a set of intermediate objects using the first model, (e.g. Fig. 5; *methods, metadata, relationships ... exposed using the infoModel*, para 0109, pg. 7; *Methods 368, BeanContext 366, typeMetaData 358* – Fig. 3A; Domain 470, 471 – Fig. 4; Fig. 12A-12D; para 0344,pg. 29 - Note: infoModel for a consumer/producer model via Mediation 112 – see Fig. 1, 13; para 0094, pg. 6 – for exposing Java methods or metadata reads on intermediate objects generated from first model, while using XIS framework for implementing of a “Domain policy usage” scenario with DSI for exposing domain descriptors and pertinent Java classes - para 0194-0198, pg. 15 – reads on using first model – see UML sequence diagram, Fig. 11; Fig. 36A-D -- to generate intermediate Java objects)

wherein the set of intermediate objects comprises Java objects (e.g. Fig. 5; para 0194-0198, pg. 15; *JavaBeans ... package com.xis.leif.event ... includes interfaces* - para 0210-0213, pg. 16; *domain policy methods ... returned ... procedural components of the metadata and methods* - para 0334, pg. 29);

generate an API using the set of intermediate objects as inputs (e.g. Add resource class, loads other pluggable services into pluggable manager – Fig. 29; para 0344, pg. 29; *Java interfaces ... instantiate the appropriate bean* -para 0349-0350 pg. 30; *exposing attributes, Java introspection* - para 0129-0135 pg. 9).

Kadel does not disclose receiving first model from a storage device. Kadel discloses implementing a "Domain Policy" using XIS framework and DSI interface to expose java objects based on metadata relationship information derived for a domain (see Fig. 13; para 0193, 0203, pg. 4; Fig. 14, 36A-B), relationship to implement the required data flow between source and consumer; and further discloses database for assisting the DSI being represented as extensible markup schema(e.g. para 0288-0304, pg. 24; para 0084-0085, pg. 5) to represent said *Domain Policy* attributes or typemetadata, attribute relationship or dependency, or declaration constraints, such that the metadata representing a application domain and Java related beans exposed by the DSI (see para 0311-0312, pg. 26) in XIS framework are represented in this XML schema form, such that the XML schema can in reverse, be **imported back** into a framework (see Kadel: para 0083, pg. 5; *XML schema ... sends to the receiver ... receiver reconstructs the information* - para 0305, pg. 25) its content exposed by the XML-DSI (para 0318, pg. 26) in relation to the database (Fig. 30). The interchangeability of XML and UML in terms of mapping of UML constructs into XML schema and vice versa was well-known and disclosed in Hu (see Hu: Fig. 6-7; para 0032-0036, pg. 3; Metadata Interchange - para 0004, pg.1). That is, such interchangeability XML/UML is also conveyed in Kadel, where a UML model constructs can be extended into a format like XML schema used as distributed application. In a meta-implementation method, Severin discloses UML constructs (Severin: Fig. 4-8) analogous to Kadel or Hu, and use of XML metasyntax and XMI support (Severin: para 0184, pg. 15) to represent meta information or constraint (e.g. Severin: zero-to-many, metahints, relationship, constraint, datatype – see para 0139-0148, pg. 11-12) for UML models which are basically founded on underlying Java classes or package (para 0196, pg. 16), wherein Severin's MVC integration of UML-based Java class/interfaces is supported by a meta repository (e.g. para 0107-0108, pg. 8) by way of reading a persisted model (para 0508, pg. 41) then mapping metamodel data to corresponding UML package (MVC para 0107-0108, pg. 8; Fig. 32). Based on the tight association between meta-information formatting in XML, XML interchangeability and data mapping with its database (see Kadel: para 0083-0084, pg. 5) and extensible metadata/schema persisted as meta information model (para 0304-0305, pg. 24-25) needed to develop application as seen in Kadel's UML representation of Domain Policies (Kadel: Fig. 11, 36C), it would have been obvious for one skill in the art at the time the invention was made to implement Kadel's DSI and XIS framework so that the first model to read from a storage is metamodel (e.g. a XML formatted model – Note: a XML format reads on a file stored in a application runtime memory) representing a first model (e.g. a domain model in terms of UML equivalent as taught in Hu – Note: a UML represented as a XML model reads on memory-stored file) as taught above in Severin's MVC's approach, because by mapping the metadata or descriptor associated with this **stored** meta-information model/file (including descriptor, constraints, relationship, type definition), derived information for UML class or Java object implementation can be obtained to support the creation of the target code, or Beans as contemplated in Kadel, in view of the portability and wide applicability of meta-information as taught in both Kadel, and Severin, whereby would reduce code translation effort from the developer using existing software reuse approach constrained by proliferation of models (see Severin: Background).

Nor does Kadel explicitly disclose API (using the set of intermediate objects as inputs) such that the API enforces the relationships and the constraints defined in the first model and enables accessing the development objects. Kadel discloses a tight relationship between the Java class underlying the model whose metadata or intermediate Java components are exposed by the mediator layer (Fig. 2, 3A, 4), and metadata descriptor and domain-related relationship (Fig. 4, 4, 9, 11) between elements of XIS Java packages (Fig. 3B; Fig. 12A-12D; Fig. 33) and the descriptor representing constraints or declarative information for a validation function (para 0153-0156, pg.11; 0167, pg. 12; value constraints - para 0178, pg. 13). Analogous to Kadel, Hu discloses UML extensible in a XML format, and like Hu, Severin discloses association of metadata describing the constraints for one to implement Java code based on such metainformation (para 0139-0148, pg. 11-12), whereby interfaces to validate and enforcing constraints and relationship is also disclosed further in Severin (Constraint Implementation, Constraint Handler, Constraint inst Fig. 33;Constraint instance – Fig 6-7; precondition, postcondition Fig. 11; Fig. 17, 19) where Java code/interface object includes "implementer" as instance for constraint validating. It would have been obvious for one skill in the art at the time the invention was made to implement the meta-information or validation function in Kadel so that the exposed Java objects are submitted (or used as inputs) into a implementation step where a API can be created (e.g. Kadel's instantiating a bean) using the Java objects and the constraint information as taught in Severin, so that Java interface objects (or APIs or bean instance) can include a functionality to validate or to enforce constraints as contemplated in both Kadel and Severin, because using the very (code construction) constraints or relationship information imposed by the metamodel as taught above(e.g. by Severin) would enable the modeled application under target integration to achieve a error-free internal infrastructure (e.g. indicating or notifying state of container or a context with respect to required or changed state of its attribute or events - see Kadel: Fig. 33F, G, H, I,) when its constituting elements fulfill the construction requirement set forth by the above metamodel.

As per claim 4, Kadel (in view of Severin) discloses wherein the first language comprises UML (see Kadel: Fig. 8, 11, 13, 36C-D; Severin: Fig. 4; para 0431 pg. 34).

As per claims 6-7, Kadel discloses wherein the first language comprises a customizable extension (e.g. Fig. 3A; addOneOfNService – Fig. 3B; Fig. 5; 36C-36D; para 0136 pg 9; Fig. 29); wherein the customizable extension is used to implement an additional feature of the API (refer to claim 1 based on addPluginService – Fig. 29).

As per claim 8, Kadel does not explicitly disclose wherein the additional feature comprises an indication of a file border. Kadel discloses API for JPanel package that operates on GUI component in terms of resizing, repainting, reshaping, paint Border, set Bounds, set Opaque (see JComponent, awtContainer, awt.Component - Fig. 33E) hence the identification of Gui file border in order to manipulate its graphic content is disclosed. And it would have been obvious for one skill in the art at the time the invention was made to implement the java libraries in view of the user manipulation, so that a feature included in the API would include a file border as set forth from the above, because this would help identify the target file upon which *awt* operation or painting methods would be defined.

As per claim 9, Kadel does not explicitly disclose wherein the API comprises a copy and paste operation. Kadel discloses XIS framework enabling editing of commands on GUI componetns, whereby the user can instantiate operation provided by the JAF API (see para 0349-0359, pg. 30; *canPaste()* Fig. 33b; *cut(clipboard)* Fig. 33c); and based on the copy-and-paste nature of the user operations to manipulate metadata attributes pertinent to a source/consumer scenario (see *cut and paste* - para 0335-0344, pg. 29) and to translate the user-customized parameters in a Java code procedure, it would have been obvious for one skill in the art at the time the invention was made to implement the XIS framework so that metadata and exposed Java classes libraries in view of JAF API (Fig. 33) are combined to support the creation of API type of operation to actually edit the attributes or manipulate exposed meta hierarchy using the standard GUI fabric, because this would constitute efficient use of metadata and reusable Java packages whose utilization would be consistent with the extensibility aspect of the XIS framework, the extensive editing role played by user (Fig. 37A-D), and the availability of JAF API as set forth above.

As per claim 23, Kadel (by virtue of the rationale of claim 1) discloses wherein the storage device is one of a storage module (schematized structures -- representing a UML model, as per Severin, Hu -- and imported into a XIS framework – see Kadel: para 0083 pg. 5- reads on XML/model document being stored in a runtime memory or file system of a framework).

As per claim 18, Kadel discloses a computer program product, tangibly embodied in an information carrier, for developing an application, the computer program product being operable to cause data processing apparatus to:

receive a data model defining development objects representing building blocks for developing the application, relationships among the development objects, and constraints for developing the application (refer to claim 1), wherein the development objects comprise Java objects (refer to claims 1, 14);

derive an API based on the data model (refer to claim 1) and use the API to perform operations on the development objects (Note: modifying procedure call parameters or attributes related to a operation/JAF commands – see para 0349-0359, pg. 30; Fig. 33b; 33c -- or pluggable service using beans reads on use of API in development – see: Add resource class, loads other pluggable services into pluggable manager – Fig. 29; para 0344, pg. 29; *Java interfaces ... instantiate the appropriate bean* -para 0349-0350 pg. 30; *exposing attributes, Java introspection* - para 0129-0135 pg. 9).

Kadel does not explicitly disclose receiving data model from a storage device. But this limitation has been addressed in claim 1 using Hu, and Severin.

Kadel does not explicitly disclose using the API being derived being such that the API enforces the relationships and the constraints defined in the first model and enables accessing the development objects; but this limitation has been addressed in claim 1.

As per claim 24, Kadel (by virtue of the rationale of claim 18) wherein the storage device is one of a storage module (refer to claim 23).

8. Claims 20-22 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kadel et al, USPubN: 2002/0184401, Hu, USPubN: 2003/0196168, and Severin, USPubN: 2005/0005251; further in view of Hejlsberg et al, USPN: 6,920,461 (hereinafter Hejlsberg)

As per claims 20-21, Kadel does not explicitly disclose operations to include:

creating a new development object as a transient object without an existing corresponding file(e.g. para 0312 pg. 26 – Note: creating of template then use it for adding/deleting Fig. 29-31; Fig. 33 – reads on transient object without a persistent file)

Kadel does not explicitly disclose modifying the transient object until the transient object is committed to a persistent file; and to destroy the transient object if a delete command is requested before the transient object is committed to a persistent file.

Kadel discloses code implemented to match SQL queries using XML elements to instantiate application to interface with database (para 0300-0310, pg. 25) where code to implement requires pluggable service and attribute conversion using JAF collaboration of classes with editing capabilities (Fig. 29-31; Fig. 33) wherein user's deleting, adding of data is effectuated via a template usage (para 0312 pg. 26) all of which data being temporary until determination to commit such implementation. Using a development application interface (analogous to Kadel) operating on layers or namespaces that expose class libraries or enumeration of related data structures or code constructs or tables (see Hejlsberg: col. 6; Fig. 2) Hejlsberg discloses application code instantiation from the libraries of reuse classes or OO packages (e.g. C++, Jscript, Microsoft ".NET" APIs) including UI objects with procedures to save a view, to customize drawing or drag-drop (col. 7, lines 48-62; col 8 lines 22-50) and a SQL namespace to interface with a database (col. 8 line 50 to col 9 line 11) including procedures to validate proper constructs, for implementing operations as to commit, dispose, rollback, save, accept/reject changes, cancel Edit (RejectChanges, acceptChanges – col 55; commit, delete, rollback – col. 57; CancelEdit col. 64; Delete, AcceptChanges – col. 65; Dispose, Finalize – col. 289; Commit, Dispose, Rollback, Save - col. 326). Based on methods based on reuse libraries to implement API in terms of constraints as in Severin and Kadel and the intended framework enabling users to decide whether how to add/remove or commit template/transient data/objects, it would have been obvious for one skill in the art at the time the invention was made to implement the code or APIs based on core libraries as practiced in both Kadel and Hejlsberg, such that a transient object in the process of validating data, information, and implementation details as taught in Kadel's user-driven customization approach (e.g. using template) would be supported by capability to create APIs with methods to destroy a transient object or to commit it to a persistent form, as taught in Hejlsberg from above. One would be motivated to do this (i.e. create APIs by the user to destroy a transient object if it is not made for persistence committing) because that way the created APIs would enable changes caused by a customization view in Kadel's approach to detect errors prior to commit, and allowing removal of undesired implementation gathering of data, whereby obviate potential runtime errors should actual translation of uncorrected constructs become finalized.

As per claim 22, Kadel does not explicitly disclose instructions to mark the persistent file as deleted if a delete command is requested after the transient object is committed to a persistent file. Based on Hejlsberg's DB-related method to indicate that change data is not accepted or that a transient form of changes is committed, or to rollback otherwise (e.g. RejectChanges, acceptChanges – col 55; commit, delete, rollback – col. 57), the notion of keeping a change with persisting of a accepted version along with removing an older version is suggested. Hence, this method as to mark an older file/record as deleted after a persisting operation has been completed (by creating a new file) would have been obvious in view of the requirement to reconcile persisted data (e.g. DB records, not keeping two records with same identification) rationale as set forth above.

9. Claims 15-17 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kadel et al, USPubN: 2002/0184401, in view of Severin, USPubN: 2005/0005251.

As per claim 15, Kadel discloses wherein the XML schema (para 0297-0306,pg. 23-25) to represent the meta information derived from the DSI and XIS framework to implement a Domain policy paradigm which is represented by UML constructs (see Fig 11, 13) hence the notion of a tree (or hierarchized organization of markup elements) based on aggregation relationships in the first model or relationship of UML constructs is evident. Based on the well-known concept that UML include aggregation relationship such as disclosed in Severin (*one-to-many, many-to-one* - para 0163, para 0223, para 0235, para 0362) it would have been obvious for one skill in the art at the time the invention was made to implement Kadel's effectuating of XML schema in view of reusing this schema into further framework (para 0083, pg. 5) so that the UML aggregation relationship of elements required for the source/consumer paradigm would be preserved and reconveyed by the hierarchized format well-known in XML as shown above, as this would help reconstructing of the constraints and relationship based on well-formatted tree organization, all of the latter being instrumental in Kadel's approach (see para 0320, pg. 27; Fig. 11, 13) wherein, for example, a state sequence in UML (see Fig. 36D) whose requirement/data dependency are re-exposed via retransforming the schema stream(see Fig. 30), such that the UML aggregation relationship thus reconstructed (data dependency in terms of one-to-many or many-to-one) would enable the XIS developer to properly implement the intended (first model source/consumer) domain sequence.

As per claims 16-17, Kadel does not explicitly disclose wherein the XML schema includes a reference based on an association relationship in the first model, and wherein the XML schema includes a complex type extension based on an inheritance relationship in the first model. UML as shown in Kadel includes association relationship and inheritance relationship (Fig. 3B; para 0080, pg. 5 para 0198-0200, pg. 15) when exposing meta-attributes related to the source/consumer model and data dependency flow. Based on Severin meta integration requiring mapping of complex association with need for new type creation (complex , new type - para 0086, 6; para 0186) among UML type hierarchies, it would have been obvious for one skill in the art at the time the invention was made to implement the XML schema intended to be reused in a XIS framework so that reference to association relationship to a UML and complex type extension are also represented in order to address the type extension and association needed within defined UML hierarchy, as by the mapping and integration (as in Severin) wherein integreting the schema would need to address complex processes requiring extension into new complex types.

10. Claims 2-3 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kadel et al, USPubN: 2002/0184401, in view of Hu, USPubN: 2003/0196168; and Severin, USPubN: 2005/0005251, further in view of Barrett et al, USPN:7,000,219(herein Barrett)

As per claims 2-3, Kadel does not explicitly disclose instructions to convert the first model to a second model in a second language, wherein the set of intermediate objects is generated using the second model, wherein the second language comprises XML. The deriving of Java classes is based on the instance of UML model (see metadata interchange in HU) and the use of metamodel to expose the amount of Java intermediate objects has been addressed in claim

1 (using Kadel, Hu, Severin). Kadel's XIS framework is disclosed as using of XML exposing interface to discover the corresponding model constructs established as a persisted XML schema (XML DSI para 0318, pg. 26) and, as a counterpart, Severin discloses XMI to rediscover content of a meta XML file (para 0184, pg. 15), hence the transforming of initial XML model into a tree or DOM elements based on the well-known W3C XMI methodology discloses transforming to a second model including XML components; e.g. creating or deriving a DOM and a token stream with regard to a schema obtained from parsing a XML file, and this is shown in Barrett's UML-based metamodel derivation with XMI adapter (Fig. 3-8; DOM – Fig. 9; col. 7-8) where the XML stream is parsed into token classes or XMI objects. Based on the UML constructs and derived class objects as taught in Kadel's use of the DSI approach and XML processor (XML stream 3002, XML Processor – Fig. 30), it would have been obvious for one skill in the art at the time the invention was made to implement Kadel's XML schema as first metamodel read **from a storage** as set forth in claim 1 (see rationale of claim 1), so that a transformation to this model yield a XML-compliant model (as a DOM using a XMI methodology) thus exposing the UML instance (see Severin) and underlying Java objects as taught above, because this second model would be used to better collect and identify objects exposed from the first model received in a portable schema stream format using the W3C methodology (see Barrett) and its useful techniques supporting this XML/model interchangeability as this is also perceived in Kadel, Hu and Severin.

11. Claims 11-12 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kadel et al, USPubN: 2002/0184401, in view of Severin, USPubN: 2005/0005251, further in view of Barrett et al, USPN:7,000,219(herein Barrett)

As per claims 11-12, Kadel does not explicitly disclose instructions to convert the first model to a second model in a second language, wherein the set of intermediate objects is generated using the second model, wherein the second language comprises XML. But based on the interchangeability of XML schema and UML constructs via using XMI as in Severin to rediscover UML constructs (Severin: para 0184, pg. 15) while Barrett discloses use of DOM creation to identified XML stream token using the W3c XMI methodology as set forth in claims 2-3. Hence, converting a imported XML schema into a framework (see Kadel: para 0083, pg. 5) would be construed a motivation as to reconstructing UML objects by using the XMI methodology, as this transforming of UML/XML stream (first model) into a DOM elements (second model having XML token) has been set forth above using the obvious rationale in claims 2-3.

12. Claim 19 is rejected under 35 U.S.C. 103(a) as being unpatentable over Kadel et al, USPubN: 2002/0184401, Hu, USPubN: 2003/0196168, and Severin, USPubN: 2005/0005251, further in view of Yeluripati et al, USPN: 7,086,065 (hereinafter Yeluripati)

As per claim 19, Kadel does not explicitly teach API including interface layer, proxy layer, and state layer. Kadel, however, discloses implementing a beans API but does not specify wherein the API comprises an interface layer, a proxy layer, and a state layer. Analogous to Kadel distributed network with use of J2EE and CORBA methodology (para 0311-0312 pg. 25-26), Yeluripati teaches Web client/server session (Fig. 8) and creation of "functional bean" based on the EJB architecture including bean state (col. 4 li. 12-42), a proxy role(Fig. 6), and acting as an interface (col. 3 li. 47-62). It would have been obvious for one skill in the art at the time the invention was made to implement the XIS-based creation of API or beans in Kadel (*Java* *interfaces ... instantiate the appropriate bean* -para 0349-0350 pg. 30) so that such API would include a *interface* layer, a *proxy* layer and a *state* layer according to the above functionalities in EJB (as being applied by Yeluripati), because as implemented in this variety, this API would include all the needed service functionalities or layers as to be able to be called upon to handle the session and interchange between client, servers and the middle services like Corba, a NW paradigm using bean container methodology which Yeluripati also endeavors.

Response to Arguments

13. Applicant's arguments filed 3/20/09 have been fully considered but they are moot in light of the new grounds of rejection which have been necessitated by the Office's reopening of the Prosecution.

Conclusion

14. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Tuan A Vu whose telephone number is (571) 272-3735. The examiner can normally be reached on 8AM-4:30PM/Mon-Fri.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Lewis Bullock can be reached on (571)272-3759.

The fax phone number for the organization where this application or proceeding is assigned is (571) 273-3735 (for non-official correspondence - please consult Examiner before using) or 571-273-8300 (for official correspondence) or redirected to customer service at 571-272-3609.

Any inquiry of a general nature or relating to the status of this application should be directed to the TC 2100 Group receptionist: 571-272-2100.

/Tuan A Vu/

Primary Examiner, Art Unit 2193

May 31, 2009